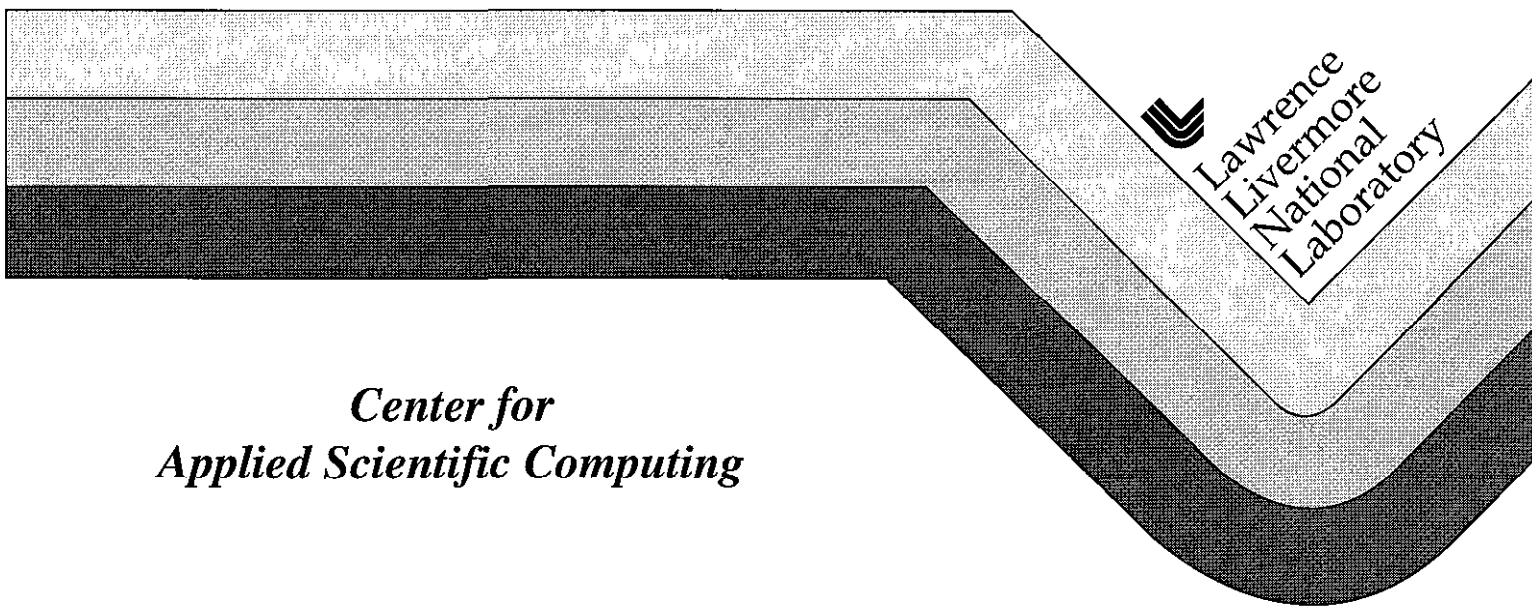
Automatic Generation of Warehouse Mediators Using an Ontology Engine

T. Critchlow
M. Ganesh
R. Musick



Center for
Applied Scientific Computing

UCRL-JC-129930 Rev. 1
April 1998

Automatic Generation of Warehouse Mediators Using an Ontology Engine

Terence Critchlow Madhavan Ganesh Ron Musick
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
{critchlow | ganesh | rmusick}@llnl gov

Abstract

Data warehouses created for dynamic scientific environments, such as genetics, face significant challenges to their long-term feasibility One of the most significant of these is the high frequency of schema evolution resulting from both technological advances and scientific insight Failure to quickly incorporate these modifications will quickly render the warehouse obsolete, yet each evolution requires significant effort to ensure the changes are correctly propagated DataFoundry utilizes a mediated warehouse architecture with an ontology infrastructure to reduce the maintenance requirements of a warehouse Among other things, the ontology is used as an information source for automatically generating mediators, the methods that transfer data between the data sources and the warehouse The identification, definition and representation of the metadata required to perform this task is a primary contribution of this work

1 Introduction

The DataFoundry research project at LLNL's Center for Applied Scientific Computing is investigating data warehousing in highly dynamic scientific environments By developing a warehouse within the rapidly evolving domain of genetics, we are able to evaluate the impact of design decisions in a realistic environment This warehouse is aimed at providing a uniform view of data obtained from several heterogeneous data sources, each containing distinct but related data from various genetics domains These data sources are primarily large, community resources whose purpose is to distribute information to the research scientists Each of these sources contains both information for which it is the sole distributor, and information that can be used to associate it with data from other sources Thus, it is impossible to completely ignore a particular data source and still provide access to all of the data While the underlying data management systems vary from flat files to OODBs, interaction with the underlying data source is primarily through form based queries Although this warehouse will greatly aid biologists by providing a single, consistent interface to this data, several significant obstacles must be overcome in order to ensure its long-term feasibility

Primary among these challenges is reducing the impact of schema modifications on warehouse availability and reliability Based upon previous efforts, we anticipate one schema modification every 2-4 weeks once all of the desired sources have been integrated Depending on how extensive the change is, directly incorporating these modifications requires explicitly modifying some combination of the wrapper program, the transformation method, and the warehouse population method Since the warehouse will remain out of date, or worse inconsistent, until the mediator is able to correctly translate data from the new representation to the desired warehouse representation, this brute force approach results in an unacceptable amount of warehouse down-time

DataFoundry is based on a mediated data warehouse architecture, in conjunction with a carefully designed ontology infrastructure, to address this problem The mediated architecture provides a balance between data cached locally for performance, and data stored remotely *Mediators* interact with data source wrappers to obtain the data contained within the source and transfer it to the warehouse while resolving semantic and syntactic conflicts The DataFoundry ontology is a practical effort to automate mediator generation, thus reducing the work required to adapt to source schema changes

This paper focuses on the use of an ontology as a formal method to store and make use of the metadata required to perform automatic mediator generation A summary of related efforts is presented next, followed by a brief discussion of mediators and their roles in managing

Proceedings of the 5th KRDB Workshop
Seattle, WA, 31-May-1998
(A Borgida, V Chaudhri, M Staudt, eds)
http //sunsite informatik rwth-aahen de/Publications/CEUR-WS/Vol-10/

schema change Section 4 presents the DataFoundry ontology and motivates the design decisions, while Section 5 outlines how it is used to generate the mediators We conclude with a discussion of possible future research directions

2 Related Work

There are two areas of ontology based research related to this work using ontologies to describe domain-specific knowledge, and using ontologies as an information resource for database integration Efforts such as Cyc [LG90], the Generalized Upper Model [BMR], the Unified Medical Language System (UMLS) [COS] and Miller's work on information capacity [MIR93] are primarily concerned with accurately representing a particular subset of domain specific knowledge This work focuses on ensuring the semantics represented in the model accurately reflect the real world semantics of domain-specific concepts Other efforts such as the Information Manifold [KLS95, LRO96, FKL97], TSIMMIS [CMH94], and InfoSleuth [BBB97] utilize ontologies as an information repository used to guide program execution In these projects the semantics of the concepts represented in the ontology, while important, are not the focus of the work Because this is similar to our approach, two of these projects, the Information Manifold and TSIMMIS, are discussed in slightly more detail

The Information Manifold uses a federated database architecture to provide a consistent view of data represented in a distributed heterogeneous environment The emphasis of the project is on correctly identifying the relevant set of data sources for a particular query To this end, an ontology is used to represent information about the contents of each data source The available information includes both the type of information available from the data source, represented as a query on the global schema, and the coverage of the source, represented as the likelihood that an arbitrary instance of the data set is present in the data source When given a query, the Information Manifold uses this information to identify the relevant data sources, and order them based on the how likely they are to contain information required by the query By querying the most relevant sources first, this approach makes effective use of limited resources

TSIMMIS takes a different approach to integrating heterogeneous data sources, providing a multi-database interface instead of a federated schema As a result, the end user must resolve the semantic and syntactic conflicts that arise between the data sources To help ease this burden, the table attributes are tagged with metadata describing the value's semantics For example, a temperature field could be marked as "degrees Fahrenheit" While this is a practical approach to accessing heterogeneous data sources, multi-databases place a heavy burden on the end-user, and are infeasible when unless all users are familiar with the semantic differences between the underlying databases

An important issue orthogonal to what information is contained within an ontology, is how that information is represented One of the major problems in the area of ontology research is the inability to transfer knowledge between different systems This arises not only because the systems represent different, possibly conflicting, aspects of a problem domain, but also because the actual data representations are incompatible This is particularly significant problem in the domain of intelligent agents, where knowledge sharing between heterogeneous systems is a critical requirement In an effort to help alleviate this problem, the Stanford Knowledge Systems Laboratory project, as part of the DARPA knowledge sharing effort, has developed Ontolingua [Gru92, FFR97], a language capable of translating between various ontology representations including the knowledge interchange format (KIF) [GSS94] and LOOM [Mac93] While not a complete solution, this effort should reduce the effort required to exchange knowledge between existing knowledge bases

As previously discussed, ontologies are a common method of representing knowledge for interacting with heterogeneous data sources DataFoundry's usage differs from these efforts in that it represents a different set of knowledge, the knowledge required to identify and resolve both syntactic and semantic conflicts between information stored in these sources The DataFoundry ontology is represented using Ontolingua because its ability to translate an ontology into several different formats provides the greatest flexibility

3 The Role of Mediators

In a mediated warehouse architecture, mediators are used to obtain data from a source and transfer it to a warehouse This data is then used to either populate warehouse tables or to dynamically answer a query, which is irrelevant for purposes of this discussion Ideally, the mediator interacts with the data source through a wrapper, which is capable of interpreting the data source and forwarding the required data to the mediator Depending on how the data source is represented (e g , relational database, flat files, etc), the wrapper may simply be the default DBMS interface, or it may also be custom built Unfortunately, when the wrapper is custom built the distinction between the wrapper and mediator may become blurred Frequently in operational databases the wrapper, transformation, and population functionality are combined into a single program, as shown in Figure 1 (a) In this situation, whenever the source schema changes, both the wrapper and the transformation methods must be manually updated to reflect these modifications This can entail a significant effort due to the complexity of the resulting code

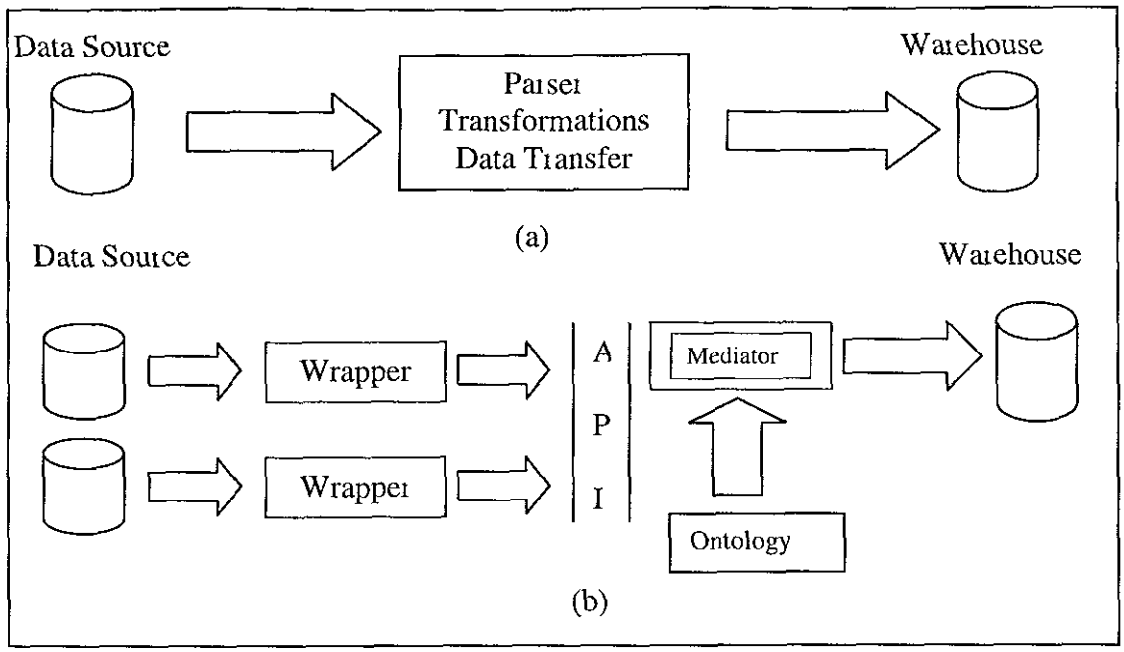


Figure 1: The Integration Process

The DataFoundry approach is shown in Figure 1 (b) In it, the wrapper is clearly separated from the mediator by a well-defined API The API utilizes a collection of classes and associated methods such as *put_position* to accept a raw version of the data from the wrapper The mediator then converts the data to the required warehouse representation before transferring it to the warehouse This API is critical to reducing the warehouse maintenance costs because it localizes the effect of schema modifications A second advantage of this approach is that the information required to create a mediator is clearly defined, and easily represented DataFoundry's ontology describes the required metadata and defines the appropriate instances An **ontology engine** (OE) is used to automatically convert these definitions into C++ code representing the API and mediators The primary contribution of this paper is the identification, definition and description of the metadata required to perform this task

4 The Ontology

The DataFoundry ontology represents four distinct concepts that are required to generate the mediators abstractions of domain specific concepts (*abstractions*), database descriptions (*databases*), mappings between a database and an abstraction (*mappings*), and transformation functions to resolve differences in representation (*transformations*) Before exploring each of these in detail, we provide a high level description of how they interact, outlined as pseudo-code in Figure 2 All of the domain specific concepts represented in the ontology are identified as *abstractions*, and the data warehouse and each data source are described as an instance of the appropriate *database* type (e g *relation-db*) *Database table attributes* are mapped to the appropriate *abstraction characteristic attributes* through *mappings* When an *abstraction* defines multiple representations for the same *characteristic attribute*, *transformation* functions are defined to convert between them

It is important to emphasize that this ontology was defined to provide the metadata required to automatically generate mediators, and does not attempt to fully represent domain specific knowledge or knowledge about the data sources As a result, our work compliments many ongoing ontology projects Where appropriate, we highlight how this research can be incorporated into our design, although current plans do not call for such integration in the near future

4.1 Abstractions

Abstractions represent knowledge about the concepts contained within the data sources being integrated They provide the extent of the domain-specific knowledge represented within the ontology, and thus available to the OE Each abstraction contains the aggregate of the information provided by the integrated data sources, including alternative representations of a particular attribute Figure 3 presents an example of the two components that combine to form an abstraction the conceptual definition and its associated characteristics

The abstraction definitions form a highly simplified ontology of domain specific concepts They identify the concepts of interest in the domain, and provide a primitive description of the interactions between them Because this

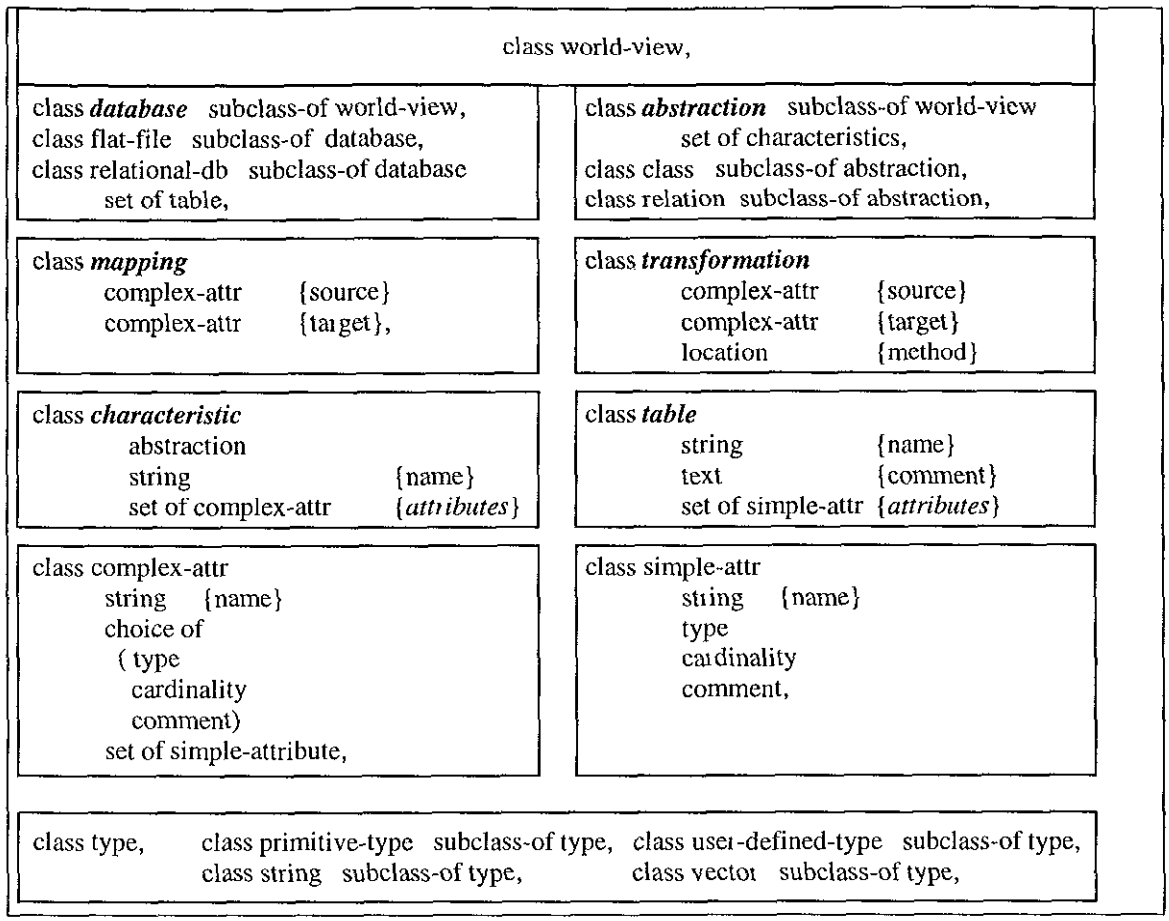


Figure 2: Outline of Ontology Constructs

representation does not include a significant amount of domain knowledge, it easy to envision expanding it to include additional information, such as mapping the concepts to the UMLS

Once the domain specific concepts have been defined, the attributes of interest need to be identified Instead of simply listing all of the relevant attributes, DataFoundry groups conceptually related attributes into *characteristics*

```
(define-class atom (?atom)
 "Atoms are the smallest elements we are
concerned with They can be combined to
form amino-acids and heterogens "
  axiom-def (subclass-of atom
                genomics)
  axiom-constraints
   (or (basic-chemical-element ?el)
       (heterogen ?el)))
```

Abstraction Defn

```
(define-instance gene-details  (detailed-rep)
 def (= gene-abs '(genomics
        ((atoms
          (id "Unique concept identifiers"
              (warehouse_key oid))
          (element "The atomic element represented by this atom"
              (short_element (string 4)))
          (position "The atoms position in 3-D space"
              (x float) (y float) (z float))
             (flexibility "How likely is the atom to move elsewhere"
              (temperature  float))
           (alternative_position "Alternate locations for the atom"
              (alternatives
                   ((x float)  (y float)  (z float)
                    (temperature float) (probability float)) N)))
```

Characteristics

Figure 3: Abstractions

```
(define-instance dw (relational-db)
  def (= dw '
        ((atom
"the atoms describe the actual position of each
atom in the current structure including het
atoms"
          (("self" oid key)
           ("model_res" oid
                  (res_in_model "self"))
           ("x" float 1)
           ("y" float 1)
           ("z" float 1)
           ("temp" float 0)
           ("element" (string 4) 1)))
```

```
(alternative_position
 "since most atoms have a single location and an
occupancy of 1 0, this table will be small relative to
the atoms table  This table defines the alternative
positions of the associated atom, and the
corresponding occupancies  The occupancy of the
original atom can be determined by subtracting the
occupancies in this table from 1 0"
          (("atom" oid (atoms "self"))
           ("x" float 1)
           ("y" float 1)
           ("z" float 1)
           ("temperature"   float)
           ("occupancy" real 1) ) )
```

Figure 4: Database Descriptions

For example, consider the *atoms* abstraction in Figure 3 Its *position* characteristic has three attributes representing its 3-D position in Cartesian coordinates If an alternative representation (e g , polar coordinates) was used in one of the data sources, it would also be included in this characteristic The decision to use characteristics to group attributes is based on our belief that this information may be useful if we investigate using the ontology to automatically define new abstraction-database mappings

4.2 Databases

As shown in Figure 4, a database description consists of language independent class definitions that closely mirror the physical layout of a relational database, the table name is followed by a list of attributes Much of this information may be automatically obtained from the metadata associated with most commercial DBMSs Unfortunately, this information must be manually entered for flat file data sources because they rarely maintain this metadata in a computer usable format Since relational databases are the most common data management system, the remainder of this paper uses relational terminology when discussing the representation It is important to remember, however, that the underlying representation is general enough to handle other architectures including OODBs and flat file collections

Figure 4 presents the two warehouse tables that correspond to the *atom* abstraction described in Section 4 1 The attribute representation used by the ontology generalizes traditional relational attributes in three important ways First, it is able to represent complex data types, as show by the *atom* abstraction *alternative_position* attribute Second, it contains arity information that defines it as required or optional and single or multi valued Finally, it provides a mechanism to define an attribute as either a key of the table or a foreign key referencing another table

The emphasis of the current research is on generating mediators to transfer data to the warehouse representation This focus requires we represent a different set of data than projects such as the Information Manifold which use source descriptions to identify the set of sources relevant to a particular query The DataFoundry representation could be expanded to include descriptions of the contents and importance of the data source if required However, because of the limited number of community data sources within the genome domain it is unlikely that this level of detail will be needed while we focus on this domain

4.3 Mappings

Mappings are used to identify the correspondence between database and abstraction attributes Because an abstraction contains aggregate information about a concept, including all alternative representations used by the data sources, there is always a direct mapping between database attributes and attributes of the corresponding abstraction Thus the basic mapping representation is a list of (construct, attribute) pairs that define this correspondence

To simplify mediator generation, these basic mappings are grouped by target construct attributes Thus a mapping is defined both from the source to the abstraction and from the abstraction to the source Because of representational differences between the databases and the abstractions, a mapping may require data from multiple classes (i e , a join) Figure 5 demonstrates how information associated with the *atom* abstraction is obtained from the warehouse *atoms*, *res_in_model* and *alternative_positions* tables, and how the *residue* and *atom* abstractions combine to describe the *atoms* table To improve readability and reduce data-

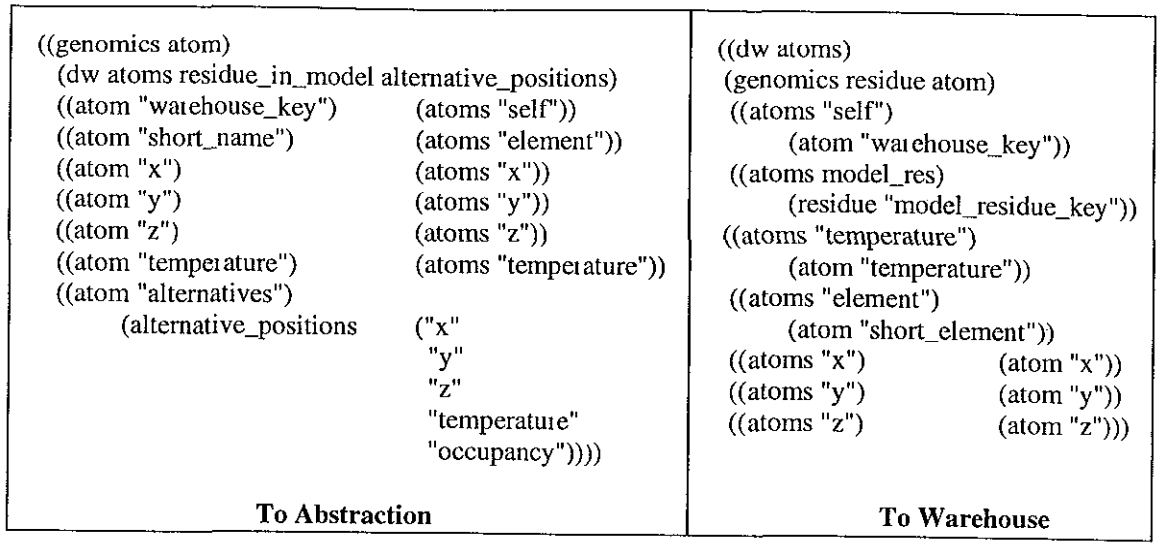


Figure 5: Mappings

entry, a short-hand version of the (construct, attribute) format is used when mapping from a single construct to a complex attribute This notation is shown in the mapping from the *alternative_positions* table to the *alternatives* attribute

4.4 Transformations

Transformations identify methods that convert between different representations of the same characteristic These methods are important in a heterogeneous environment where data representations vary widely because the source and warehouse representations are likely to be different By including the names and locations of these functions in the ontology, the OE is able to use them to resolve representational differences between the data source and warehouse automatically The OE must be able to identify the attributes affected by each of the methods in order to identify the appropriate set of transformations to apply The obvious approach is to explicitly associate the attributes with the method, however, to reduce complexity and improve readability, we have chosen to use naming conventions instead This greatly simplifies both the data entry and the mediator generation task by removing parameters from the method call while ensuring the source and target attributes remain easily identifiable

As shown in Figure 6, the ontology also allows the user to define additional methods and data structures The format used to represent these extensions is

```
(define-instance gene-transforms
        (abstraction-enhancement)
  def (= genome-transformations
     '("/DF/ontology/lib/genome lib"
        (amino_acid
           (translation-methods
              (full_to_one_char)
              (full_to_three_char)
              (one_char_to_full)
              (three_char_to_full))
           (class-methods
              (three_char_to_one_char  (three-char (string 3)) (one_char character)))
           (class-data
              ((name_conversion_table
                 ((one_char character)
                  (three_char (string 3))
                  (full_name (string 40))) 28)
                 ({ {"A", "ALA", "Alanine"}, {"R", "ARG", "Arginine"},  } ))))
```

Figure 6: Transformations

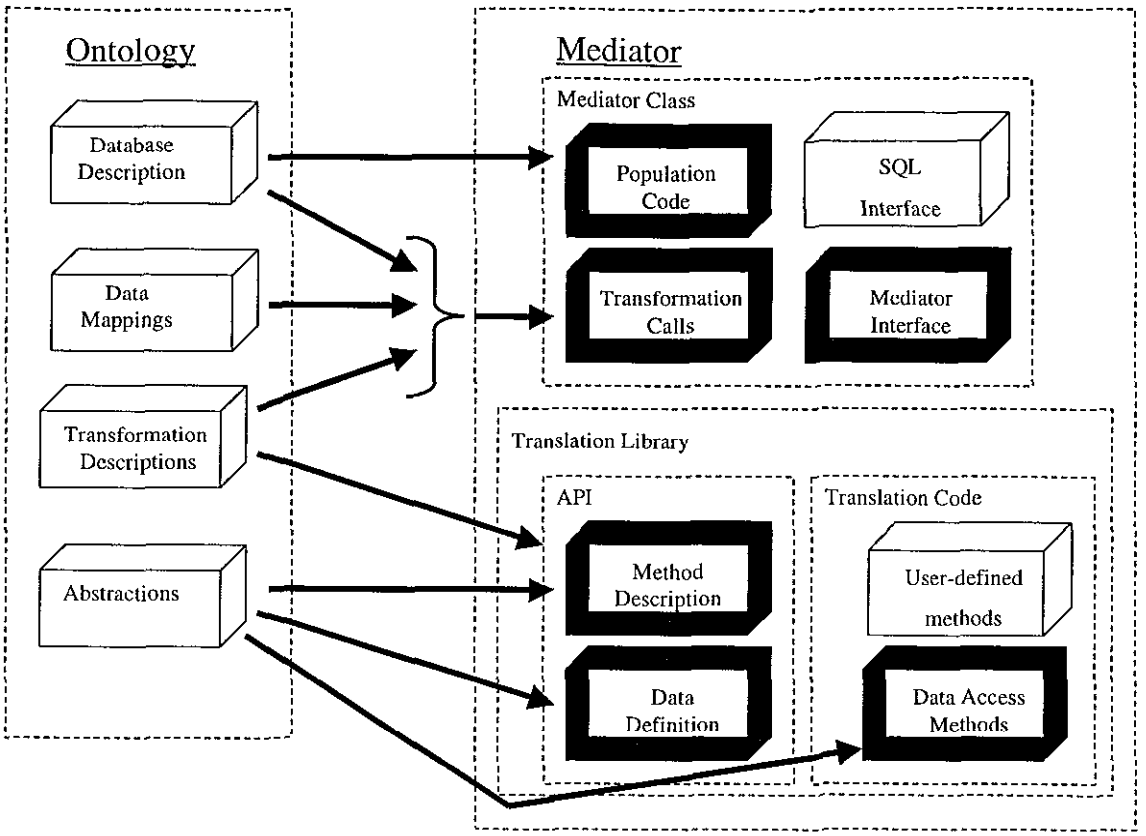


Figure 7: The Ontology and Mediator

straightforward The methods are defined by their signature, while the data structures contain the associated name, attribute list and an initial value While these extensions are not used by the OE, they provide additional flexibility to the transformation method For example, a *chain* may define several formats for a characteristic representing *sequence* of amino acids The ability to call *amino_acid* methods, such as *three_char_to_one_char*, without needing to create instance of the class greatly simplifies writing this function

5 Generating the Mediator

Figure 7 outlines how the information expressed in the ontology relates to the various components of the mediator The entire interface and the majority of code will be automatically generated from the ontology using the OE – these components are shown with a solid frame The mediator functionality is decomposed into two components the translation library and the mediator class The API available to the parser is a combination of the individual APIs The translation library is a C++ library containing a set of classes corresponding to, and automatically derived from, the abstractions, of course, the transformation methods must be explicitly entered The mediator uses the abstraction classes and mapping ontology to perform the transformation from the input data format, obtained from the parser, to the warehouse representation, as described by the ontology While converting data to the target representation may require multiple steps (based on the methods available) the naming convention makes this a relatively straightforward search process Once the data transformation has been performed, the SQL interface is used to load the data into the warehouse

Incorporating a new data source requires the DBA to describe the data source, map the source attributes to corresponding abstraction attributes, ensure that all applicable transformation methods are defined, and create the parser The OE then creates the new mediator class, and expands the data class API if needed Once a database has been integrated, adapting to minor source schema changes often requires only modifying the wrapper to read the new format Significant changes in the data representation may require the ontology to be modified and a new mediator created

6 Conclusion

In a dynamic scientific environment, maintaining the consistency and availability of a data warehouse requires quickly adapting to changes in the source schemata Effectively reducing the impact of schema changes requires addressing this issue at all levels of the architecture Our extensive use of an ontology infrastructure is expected to dramatically reduce the effort required to generate mediators, overcoming a significant obstacle Once the OE is completed and fully tested, we intend to investigate additional uses for the ontology Possible future directions include creating an ontology interface to the warehouse and integrated data sources, possibly incorporating existing research such as [LRO97], and using the ontology as a collection of hints to automate mapping definition when integrating new data sources, similar to the hints used in [Cri97]

References

[BBB97] R J Bayardo Jr W Bohrer, R Brice, A Cichocki, J Fowler, A Helal, V Kashyap, T Ksiezyk, G Martin, M nodine, M Rashid, M Rusinkiewicz, R Shea, C Unnikrishnan, A Unruh, and D Woelk InfoSleuth Agent-Based Semantic Integration of Information in Open and Dynamic Environments In *Proceedings of the 1997 ACM SIGMOD International conference on Management of Data* May 1997

[BMR] J Bateman, B Magnini, F Rinaldi The Generalized Italian, German, Enlish Upper Model In *Proceedings of the ECAI94 Workshop Comparison of Implemented Ontologies* 1994

[CMH94] S Chawathe, H Garcia-Molina, J Hammer, K Ireland, Y Papkoastantinou, J Ullman, J Widom The TSIMMIS Project Integration of Heterogeneous Information Sources In *Proceedings of the ISPJ Conference* 1994

[COS] K E Campbell, D E Oliver, E H Shortliffe The Unified Medical Language System Toward a Collaborative Approach For Solving Terminological Problems submitted to *Journal of American Medical Informatics Association*

[Cri97] T Critchlow Schema Coercion Using Database Meta-Information to Facilitate Data Transfer Ph D Dissertation University of Utah Technical Report CSTD-97-003 June 1997

[FKL97] D Florescu, D Koller, A Levy Using Probabilistic Information in Data Integration In *Proceedings of the 23rd International Conference on Very Large Data Bases* 1997

[FFR97] A Farquhar, R Fikes, J Rice Tools for Assembling Modular Ontologies in Ontolingua Stanford Knowledge Systems Laboratory Tech Report KSL-97-03 1997

[Gru92] T Gruber Ontolingua A Mechanism to Support Portable Ontologies Stanford Knowledge Systems Laboratory Tech Report KSL-91-66 November 1992

[GSS94] M R Geneseteth, M P Singh, M A Syed A Distributed Anonymous Approach to Software Interoperation Available from logic stanford edu/sharing/knowledge Nov 1994

[KLS95] T Kirk, A Levy, Y Sagiv, D Srivastava The Information Manifold In *Working Notes of the AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments* 1995

[LG90] D B Lenat, R V Guha Building Large Knowledge-Based Systems Representation and Inference in the Cyc Project Addison-Wesley Reading MA 1990

[LRO96] A Y Levy, A Rajaraman, J J Ordille Querying Heterogeneous Information Sources Using Source Descriptions In *Proceedings of the 22nd International Conference on Very Large Data Bases* 1996

[Mac93] R M MacGregor Representing Reified Relations in Loom In *Journal of Experimental and Theoretical Artificial Intelligence* 5 179-183 1993

[MIR93] R J Miller, Y E Ioannidis, and R Ramakrishnan The Use of Information Capacity in Schema Integration and Translation In *Proceedings of the 19th International Conference on Very Large Data Bases*, pages 120-133, 1993

Work performed under the auspices of the U S DOE by LLNL under contract No W-7405-ENG-48